Software and Systems Modeling (2023) 22:331–349
https://doi.org/10.1007/s10270-022-00998-y

REGULAR PAPER

# Extracting LPL privacy policy purposes from annotated web service source code

Kalle Hjerppe[1] · Jukka Ruohonen[1] · Ville Leppänen[1]

Received: 1 December 2020 / Revised: 8 December 2021 / Accepted: 7 March 2022 / Published online: 8 April 2022


**Abstract**
Privacy policies are a mechanism used to inform users of the World Wide Web about the processing of their personal data. Such processing has special requirements, since personal data are regulated by data protection legislation. For example, a consent or another legal basis is typically needed. Privacy policies are documents used, among other things, to inform the data subject about processing of their personal data. These are formally represented by privacy languages. In this paper, we present a technique for constructing Layered Privacy Language policy data from web service code bases. Theoretically, we model the purposes of processing within web services by extending the privacy language with composition. We also present a formal analysis method for generating privacy policy purposes from the source code of web services. Furthermore, as a practical contribution, we present a static analysis tool that implements the theoretical solution. Finally, we report a brief case study for validating the tool

**Keywords** Data protection · privacy engineering · privacy language · static analysis · semantic web · GDPR

## 1 Introduction

The fundamental right to privacy is included in the Universal Declaration of Human Rights adopted by the United Nations General Assembly in 1948 [4]. Despite this right, in practice privacy has been threatened by many factors, and the age of the information economy has only intensified the threats. To abate them, the right to privacy has also been portrayed as a right to the protection of individuals' personal data. This viewpoint has been prominent in Europe. In fact, the right to the protection of personal data was included in the Charter of Fundamental Rights of the European Union, adopted with the Treaty of Lisbon in 2009. The General Data Protection Regulation (GDPR) [54] builds upon this fundamental European foundation and sets the context of the present work.

Communicated by Alfonso Pierantonio.

✉ Kalle Hjerppe
kphjer@utu.fi

Jukka Ruohonen
juanruo@utu.fi

Ville Leppänen
ville.leppanen@utu.fi

[1] Faculty of Technology, University of Turku, Turku, Finland

While personal data must be protected, the capability to process it is a valuable asset for enterprises. Given this backdrop, it can be said that there is an inherent conflict between business interests and the right to privacy and data protection. However, advances in data protection can also provide new opportunities for businesses [51], although these opportunities have not yet fully materialized. One possible explanation for this is the long-standing gap between research and practice in the domain of privacy and data protection [41]. The same argument can be made about information security, which, likewise, has a long research lineage in computer science. Privacy engineering has emerged as a field that seeks to narrow this gap. In general, privacy engineering can be defined as a "*field of research and practice that designs, implements, adapts, and evaluates theories, methods, techniques, and tools to systematically capture and address privacy issues when developing sociotechnical systems*" [29]. The results from privacy engineering research can lower the adoption costs for industry and eventually bridge the gap between practice and research. Within this research domain, privacy-friendliness of a system can be understood to range from "*privacy-by-policy*" to "*privacy-by-architecture*" [50]. The former approach can be summarized as the implementation of privacy mechanisms to the extent that the users (and governance) are satisfied, whereas the latter refers to design-

ing a system in such a way that privacy is inherent to the system. This paper can be positioned in the middle: A hybrid approach, as described by Spiekermann & Cranor [50], is adopted; the goal is to make technical advances in order to improve the enforcement of privacy and data protection.

Privacy policies are necessary for many software engineering practitioners. The GDPR requires that those processing personal data must keep a record of their data processing activities, and to establish a lawful basis for each purpose of processing [54]. Such bookkeeping necessitates the documentation of the processing activities in privacy policies. Documentation is required also for establishing the lawfulness of a given processing activity and for accountability. In the Web, the consent of data subjects is presumably the most commonly used way to establish the lawful basis of processing personal data under the GDPR [18]. But to obtain an informed and freely given consent, a data subject must know and understand the purpose(s) of processing his or her personal data. Given that privacy policies have been repeatedly shown to be difficult and time-consuming to read and understand [38], it is understandable why the whole concept of consent has long been recognized as problematic in the digital era [9,11,28]. There are many practical reasons behind the problems. For one, there exists an *information asymmetry* between service providers and data subjects [7,49]. A service provider knows exactly how their service operates, while the data subject only knows information shared with them. This asymmetry motivates developing new methods for expressing privacy policies in more transparent and understandable ways. At the same time, value can be provided to service providers in terms of tool-assisted privacy policy management, and our work aims to be useful for both.

There is a continuing challenge in the current software industry: Privacy policies are neither explicit nor specific enough [42]. In order to overcome this challenge, our approach makes use of a notion that an abstract purpose for processing can often be described more specifically by asking a simple question: *"what does the purpose consist of?"* This question can then be repeated until the desired level of specificity is reached. As an illustration, consider a Web Service (WS) that processes personal data in a context of booking a hotel room online. In this context, *"hotel room reservation"* might denote the high-level purpose for the processing. However, this purpose does not provide enough details to obtain an informed consent. Therefore, the purpose could be augmented to contain more fine-grained elements, such as *"authentication," "review display," "account management," "reservation confirmation,"* and *"online payment."* In a typical WS implementation, these specific elements map to particular software architecture modules in a service-oriented architecture (SOA). By further increasing the granularity and lowering the level of abstraction, it is possible to enumerate these elements as a sequence of WS-specific functional calls. By using a formal privacy policy language, the enumeration can be further used to generate policies that describe the distinct elements in the processing. Finally, the lowest-level sub-purposes can act as building blocks to form a privacy policy. In essence, a privacy policy for a web service is also a documentation of its behavior concerning personal data processing (along with additional factors, of course).

It should be noted that the previous example only covers processing done by a primary service provider. In practice, transfers of data to other domains often occur; a single policy does not cover the entire transitive closure in the flow of personal data. In the context of the Web, there has been a culture of sharing data across businesses. Personal data are a key resource in the information economy, but with resources come the risks of data breaches, leaks, and misuse [31]. At a societal level, the risks are increasingly managed by regulations and standards that seek to protect the fundamental rights to privacy and data protection. For instance, a payment transaction might use a third-party provider. In this kind of a situation, the original privacy policy mentions the transfer and refers to their privacy policy, layering them in stack.

In terms of privacy engineering, the previous illustration underlines the possibility to switch between a *user-focused* perspective of *validating* a privacy policy against a concrete software architecture, and a *developer-oriented* perspective of *generating* the policy automatically [33]. A tool that can carry out such switching facilitates the writing of privacy policies—even though human intervention is still required in practice. The effort of compiling the policy can be assisted with automatically generated data. The current way to construct policies with privacy languages is by hand; which is rather impractical. Indeed, a large portion of privacy policies are not specified in a formal way at all [23], presumably because it is a daunting task and not required by law (yet).

With this motivation in mind, the present work lays down the groundwork for generative programming for privacy policies. The goal of this work is to formulate a method to extract data from web services in a way that fits into a privacy policy. The following two research questions (RQs) are examined:

- RQ1: *How can we model personal data processing for web services using LPL?*
- RQ2: *How to automatically extract the model's data from web service source code?*

The answers to these two questions establish both a *method* (RQ1) and a *tool* (RQ2). In terms of RQ1, existing models for privacy policies are extended and augmented in the web application context; in terms of the RQ2, a concrete implementation is presented. From a practical privacy engineering viewpoint, it is important that the results are possible to integrate into real-world applications with minimal

overhead costs. Therefore, a case study is further presented to demonstrate practical applicability.

Our contribution to the state of the art is combining both the advances in privacy languages and static analysis to improve the intersection in the context of web services. We present improvements to the theory, and a novel methodology (both in theory and practice). This paper is an extended version of our previous work presented in the 2020 International Workshop on Privacy Engineering [34], and this paper represents the final results of the research. The previous work is extended to complete previously out-of-scope requirements, with additional theory and extended experimental parts. The contribution to the model is the integration of data transfers and data recipients, and updating the LPL to the latest standard. The analysis tool is updated according to the new model, and is extended with a visualization feature that creates user-friendly HTML from the raw data of the analysis. Finally, this paper also contributes further evaluation, repeating the case study analysis with the updated model, and more comprehensive discussion.

The remainder of the paper is structured as follows. The opening Sect. 2 outlines the background and related work. The subsequent Sect. 3 provides the formal definitions for the method and the static analysis tool thereto. These definitions are provided in two parts: The LPL is first extended to accommodate composition, after which the language's *Purposes* are mapped to web services. Thus, this section provides the answer to RQ1. For answering RQ2, Sect. 4 presents a concrete implementation based on static analysis and SOAs. The use of the tool implemented is further elaborated in Sect. 5 with a small case study. Finally, Sect. 6 discusses the implications of our findings and their integration into engineering practice, suggests some preliminary quality measures, and pinpoints opportunities for further work.

## 2 Background and related work

### 2.1 The GDPR and the web

The GDPR came into force in 2018. Research concerning the regulation and technical solutions to its requirements has been active before and thereafter. The regulation covers the processing of all personal data about people of the European Union, and the changes from previous legislation obligated businesses and organizations to review their means of processing personal data. In other words, there has been both organizational and technical consequences from the GDPR [57]. Data protection impact assessments are a good example; these must be performed, for example, when new technologies are adopted for the processing of personal data. These assessments provide a way to demonstrate compliance, serving as an early warning system about risks [12]. The more explicit documentation about a system there is, the easier and lighter the assessment of the system. Indirectly, impact assessments thus provide a further motivation for the present work.

Technical approaches for GDPR compliance are as numerous as the technical ways to process personal data. Previous research has studied the requirements [5,45], implementation in software architectures [32], and enterprise modeling solutions [13], to name three examples. However, the gap between research and practice still persists. For instance—in accordance with the topic of this paper, there is an acute need for better privacy policies in the Web. Matte et al. studied the legal bases of purposes in the advertisement space [42]. Their conclusion was that several purposes were neither explicit nor specific enough to be compliant with the GDPR. For instance, they concluded that the legal basis could not be derived from a purpose called "Personalization." The apparent vagueness seemed to imply that the purpose in fact bundled multiple purposes. Again, this is an example of the information asymmetry between providers and consumers online. In this paper, we show how such bundling can be improved with privacy languages; composition of purposes makes it *explicit* what is contained in any given purpose. The ability to create fine-grained composites improves specificity for each individual purpose.

Much responsibility in the privacy engineering discipline is on the shoulders of the software engineers, but this is a challenge for them due to various reasons [10,48]. We take this as a call to action to make the "developer experience" of data protection better.

### 2.2 Privacy languages and the LPL

Multiple competing formal privacy languages have been presented in recent years. Of these, the LPL is a good and timely example as it has been explicitly designed to address the GDPR's requirements [25,26]. It adheres to both legal and technical privacy viewpoints. The main design requirements for the language included the differentiation of data subjects and data recipients, purpose-based policies for data, retention and anonymization elements, the ability to layer policies for provenance, and human readability. While some previous research exists for programmable privacy languages [44], the LPL's abstraction level focuses on the modeling of privacy policies. This level is also suitable for the present work, since it is possible to use our method with usual programs. The challenge for the type of privacy languages which integrate completely into the programming language is adoption in practice. This is the balance between holistic *privacy-by-architecture* and the "easy" method of *privacy-by-policy*. Our view, and the goal of this paper, is that there is room in between—room to improve existing software design practices with regards to data protection.

After the LPL's initiation in 2018, further effort has been devoted to extending the language and building features around it. Some examples include authentication [61], personalization [27], and privacy icons [24]. The language was updated in a later publication [25]. However, there appears to be no previous work for mapping LPL specifically to web services, nor does there seem to exist previous research for grouping purposes. Although LPL itself features a $PurposeHierarchy$, this feature is an inheritance relationship. In contrast, the present work operates with composition. The difference is significant in the practice of modeling policies, which is demonstrated later on.

One goal of formal languages for expressing privacy policies is to make them machine-readable. This readability enables algorithmic validation of a privacy policy, of which there also exists some previous work in this regard. For instance, a language has been developed for the requirements of the GDPR [14], and another for system design [58]. Compared to the original, unmodified LPL, the ontology of the language is more expressive, modeling consent, processing of data, location, and related data protection characteristics. If the goal is *compliance checking* instead of mere *formulation* of a policy, such modeling is also necessary. Against this backdrop, the present work builds upon LPL and takes it a step closer to implementation. That said, the concepts presented (excluding Sect. 3.1) are applicable to other policy languages as well.

*Access control policies* are policy languages which are grouped as security-focused in the categorization by Gerl [26]. The purpose of access policies is to regulate the access to data with rules and effects. These include examples such as *XACML* [52] and *ConSpec* [1]. Generally, these languages are defined in a generalist way, and not specifically for GDPR needs; see Gerl for comparison [26]. On the other hand, for instance, an extension of XACML with privacy awareness has been presented [3] with further similar approaches [39]. LPL lacks the kind of infrastructure support the standard-based protocols have, but the present work is a step in the direction. The notion that purposes can be seen as a directed acyclic graph which can be used for access control in general has been explored as well [64].

In essence, the GPDR entails six aspects for privacy policies: the purposes of data processing, the data processed, the potential data recipients, transfers of the data, erasure conditions for the data, and information about the processing itself [14,26]. In addition, a privacy language should be human-readable and model provenance [26]. In its current state, LPL covers these aspects, but it is cumbersome to model details about the processing in a transparent way. Therefore, the present work provides further means to document web services in detail and to generate this data automatically.

### 2.3 Static analysis

Previous research has discussed using static analysis as a technical measure to ensure data protection [33]. The term *static source code analysis* refers to the practice of inspecting the behavior of a program without executing it. In contrast, dynamic analysis monitors the behavior while the program is running. In static source code analysis, the model under inspection is the program specification or an abstract model of it. While the concept of static analysis can in general be applied to other models as well, such as validating BPMN for requirement conflicts [43], this paper focuses on models derived from source code. That said, comparable approaches toward annotating business processes in order to analyze their privacy properties also exist [8].

Static analysis is commonly used to improve software quality as it is being developed. A good example is the class of analyzers called "linters," which validate that the source code follows pre-defined style and quality rules [55]. On a higher abstraction level, the way data flow in the abstract syntax tree of a program can be assessed for security vulnerabilities [46]. Furthermore, there is plenty of research about detecting privacy leaks using static analysis as well; see [21] as an example in this regard.

This paper takes another perspective on static analysis. Based on previous work [33], static analysis can be divided into *user-oriented* and *developer-focused* approaches. The user-focused approach seeks to answer whether a program follows external rules (i.e., a policy). Often the use case is indeed that a customer analyzes a program to see whether its publishers are honest. In contrast, the developer-focused approach describes the program and creates documentation from the structure of the program. Although generating an entire privacy policy is not simple in practice, this approach seeks to remedy possible disconnects between a policy and behavior.

### 2.4 Privacy in web services

In the web service literature, plenty of different modeling and analysis tools have been presented. The so-called semantic web would be a prime example [2,56,59]. There are many industry tools and standards for documenting Application Programming Interfaces (APIs), of which a good example would be the OpenAPI initiative.[1] In general, these API frameworks provide means to describe the functionality of a service and to increase its discoverability [20]. The OpenAPI specification has been extended with metadata in order to improve interoperability [17,63]. Discoverability and interoperability align with the GDPR's high-level goals. Salnitri et al. have modeled privacy and security requirements of socio-

[1] https://www.openapis.org/.

technical systems in general [47]. The notion of modeling privacy in web services with composition is a known concept as well [62].

The Platform for Privacy Preferences Project (P3P) was an effort to integrate machine-readable privacy policies to the Web [15,36]. Alas, the standard has been deprecated[2], allegedly due to the lack of support in web browsers. After the P3P, there have been some follow-up projects, including the so-called Policy Aware Web [60]. New initiatives have also been presented in the industry, for instance, the Interactive Advertising Bureau have pushed a so-called Transparency & Consent Framework (TCF)[3] in an effort to reconcile targeted advertising with the GDPR. Despite the open issues raised [42], it has become popular. The popularity supposedly demonstrates the difficulties in the adoption of privacy enhancing technologies. In any case, the adoption of technical solutions is a factor that must be taken into account in the present work.

Generating API documentation from source code is an industry standard practice, and there are tools that integrate this task with OpenAPI.[4] In addition to documentation, these generators can create client libraries that use the APIs. Another approach is to use business process modeling and existing enterprise architecture documentation to generate specific GDPR-related documentation [35]. Such an approach has its merits—yet it is decoupled from the technical implementation of services.

Despite these advances, there appears to be no existing tools specifically tailored for generating privacy policies from source code. The state of the art in privacy languages is branched, featuring either entirely theoretical languages or languages with an application fixed to a certain technology. The previous version of our work [34] presented the foundation for this goal, but it was incomplete. By building upon an existing solution for annotating personal data processing in source code [33], this paper narrows this apparent gap in both research and practice. Our contribution combines the advances in privacy languages and static analysis to improve the state of the art in their intersection.

### 2.5 Petri nets

The chosen approach to modeling web services in this work is based on Petri nets, as introduced by Hamadi and Benatallah [30], who provide an algebra that is capable of composition and represents the way web services function in practice. Petri nets themselves are a visual modeling technique based on directed graphs of nodes and places [30]. This perspective on web services is suitable for modeling combinations of different services, and the parallel to compositions of layered privacy policies is apparent. On this topic there are further related work as well—for example, Diver and Schafer make use of the modeling technique in order to visualize information flows within applications in order to facilitate privacy by design [19].

## 3 Definitions

This section defines concepts of our approach to the research questions. The first definition is a composed $Purpose$, an extension to LPL that is accompanied by related constraints. This composition is used as a bridge from the formal policy language level to the technical implementation level. The composition is subsequently mapped as a directed graph to the structure of web services represented as a Petri net. While most of the definitions appear in the literature, the forthcoming discussion completes them with additional constraints and extensions for $Data$ transfers and recipients. We include the entire extensions of LPL (from [34] and new) here for the sake of completeness. Whereas previous work is based on an earlier version of LPL [26], the definitions here are consistent with the current version [25].

The LPL *Purpose* "denotes a reason and extent of the processing of personal data." [25] and matches to the GDPR Article 5 *Purpose of Processing*. This is the foundation upon which privacy policies are built. The subsequent sub-purposes, on the other hand, are not supposed to be standalone. Rather, they can bridge the abstract legal purposes into practice by specifying their component parts.

### 3.1 Defining *purpose* composition

The LPL has a fundamentally layered nature, as indicated already by its name. The layering is used to provide a simplified overview while preserving the details for interested parties. The mechanisms of layering in LPL are nested privacy policies with $UnderlyingPrivacyPolicy$ and an inheritance mechanism for $Purposes$. While these are useful, our application requires another way to layer granularity and abstraction levels into the language, to reason about the component parts that a $Purpose$ consists of. In this context, *inheritance* is understood as an *is-a* relationship—a child may be substituted for its parent—whereas *Composition* refers to a *has-a* relationship of holding a reference to another component element. The LPL forbids multiple inheritance, but composition has no such restriction.

To this end, we will define a relation of $Composed\ Purposes$, but let us start with the definition of a policy. A $Layered\ PrivacyPolicy$, or $lpp$, that is, an element representing a

[2] https://www.w3.org/TR/P3P11/.

[3] https://iabeurope.eu/transparency-consent-framework/.

[4] https://swagger.io/tools/swagger-codegen/.

privacy policy, is defined [25, Def. 5.6] as a tuple:

$$lpp = (version, name, lang, ppURI, \widehat{HEAD}, \widehat{DESC}, \widehat{I}, ds, \widehat{P}, \widehat{C}, \widehat{DPO}, dsr, lc, upp), \tag{1}$$

where $version$ describes the LPL version, $name$ labels the given policy, $lang$ specifies the language with which the policy is written, $ppURI$ is a link to a textual version of the policy, and $\widehat{HEAD}$ and $\widehat{DESC}$ are reference elements of human-readable information. $\widehat{I}$ refers to a set of Icons, $ds$ to the *DataSource* of the policy in question, $upp$ is the $UnderlyingPrivacyPolicy$ element, and $\widehat{P}$ is the set of $Purposes$ the policy consists of. Furthermore, $\widehat{C}$, $\widehat{DPO}$, $dsr$, and $lc$ define information about the controller, data protection officer, data subject rights, and instructions to lodge complaints, respectively. Of these, $upp$—*UnderlyingPrivacyPolicy*—allows composing policies into layers, while $\widehat{P}$ defines the actual policy content. An LPL $Purpose$ element, $p$, is defined [25, Def. 5.17] as the following tuple of values:

$$p = (name, optOut, required, pointOfAcceptance, \widehat{HEAD}, \widehat{DESC}, \widehat{D}, \widehat{PM}, \widehat{PSM}, \widehat{DR}, \widehat{LB}, \widehat{ADM}, r), \tag{2}$$

where *name* defines the identifying name of the purpose, and the Boolean values $optOut$ and $required$ define whether the purpose has to be "actively denied" and whether the purpose has to be accepted to be able to consent to the policy. $\widehat{HEAD}$ and $\widehat{DESC}$, again, are reference elements of human-readable information. Furthermore, $\widehat{D}$ refers to the set of $Data$ elements accessed for the given $Purpose$, and $\widehat{DR}$ the set of $DataRecipients$ the right to access is granted. $\widehat{PM}$ and $\widehat{PSM}$ refer to applied de-identification methods: *PrivacyModels* and *PseudonymizationMethods*. These specify the privacy conditions which have to apply to the Data elements, such as *k-anonymity* [25]. Finally, $\widehat{LB}$ defines the legal basis of the purpose, $\widehat{ADM}$ whether the purpose contains automated decision making or profiling, and $r$ refers to its retention rules.

Given these preliminaries from the LPL definitions, our goal is to extend the language by defining the relation among composite $Purposes$ in a $Policy$. Since a root-level $Purpose$ in a privacy policy is often not specific or concrete enough to map to a technical implementation unit, the hierarchy between $Purposes$ can be defined as a directed graph. However, abstract concept-level purposes (e.g., "reserve a room") contain many concrete parts (e.g., API calls). Therefore, the aim is to also define constraints under which data subjects accepting a given $p$ may accept all its composite purposes, denoted by $p'$, without weakening their data protection. The relation is specific to a $Policy$, which defines the hierarchy—policies may have different hierarchies.

Let $up$ denote a relation for an $UnderlyingPurpose$, represented as a tuple of values in a policy, and $\widehat{up}$ a set of $up$. Further denote the relation of all $ComposedPurposes$ in an $lpp$ by $cp$. Then,

$$up = (p, p') \quad \text{and} \quad cp = (lpp, \widehat{up}), \tag{3}$$

where $p$ is any $Purpose$ in the policy, and $p'$ is a $Purpose$ element with additional constraints to form a valid privacy policy, as follows. From the relationship that a composite $p'$ is a component of a $Purpose$ $p$, the constraints for a $Policy$ can be reasoned. In any given $lpp$, where $p$ is a $Purpose$ belonging to it and $p'$ its underlying $Purpose$, i.e., the tuples in (3) exist and $up \in \widehat{up}$, in order for the $lpp$ to be valid, it must hold that

$$\widehat{D}_{p'} \subseteq \widehat{D}_p, \tag{4}$$
$$\widehat{DR}_{p'} \subseteq \widehat{DR}_p, \tag{5}$$
$$r_{p'} \leq r_p, \tag{6}$$
$$\widehat{PM}_{p'} \geq \widehat{PM}_p, \tag{7}$$
$$\widehat{PSM}_{p'} \geq \widehat{PSM}_p, \tag{8}$$
$$required_{p'} = required_p, \tag{9}$$
$$optOut_{p'} = optOut_p, \tag{10}$$
$$\widehat{LB}_{p'} \subseteq \widehat{LB}_p, \tag{11}$$
$$\widehat{ADM}_{p'} \subseteq \widehat{ADM}_p. \tag{12}$$

In (6), the inequality is defined as a strictness of a *Retention* element depending on its particular $type$. Let $Indefinite \geq AfterPurpose \geq FixedDate$ and assume that comparisons with the same $type$ are decided by $pointInTime$. With these assumptions, $AfterPurpose \geq FixedDate$ follows from not considering a $Purpose$ completed until all $FixedDate$ retentions are resolved. It is also worth remarking that the comparison of $PrivacyModel$ elements in (7) is not defined strictly. The reason originates from the LPL, which does not provide a rigid definition for a set of potential (pseudo)anonymization methods. A comparison of two $pm$'s with the same $name$ (e.g., "*k-anonymity*") is decided by the values of their $PrivacyModelAttributes$. Without a definition of an exact comparison across different $pm$ types, it is possible to avoid the issue by augmenting the criteria with a requirement that the $type$s of $pm_{p'}$ and $pm_p$ must be equal. The criteria in (9) are conservatively defined for now: It can be argued that $required_{p'} \geq required_p$ is sufficient [37], but that raises further questions as well and is not as unambiguously backwards-compatible.

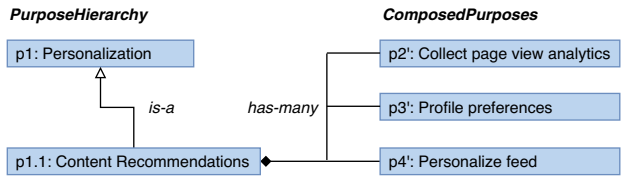


**Fig. 1** An example of a possible *PurposeHierarchy* using *ComposedPurposes*, highlighting both composition and inheritance. Re-drawn after [34]

The definition for $ComposedPurposes$, $cp$, accompanies another structure originally defined in the LPL that is named $PurposeHierarchy$, $ph$. $PurposeHierarchy$ defines a parent–child relationship as an inheritance hierarchy. While this hierarchical *is-a* relationship is similar to the *has-many* relationship for $ComposedPurposes$, there exists a meaningful difference in capability and intent. In other words, the relationships are complementary. Whereas inheritance enables reusing rights and making a $Purpose$ concrete, $ComposedPurposes$ expose the actual contents of the $Purpose$. Following the Liskov substitution principle [40], inherited *Purposes* ought to behave like their parent. In contrast, composed *Purposes* do not share this limitation and can thus more explicitly represent a part of the parent. The following brief example can be used to elaborate this point further.

Consider "personalization" as a parent $Purpose$ (and Fig. 1 for a visual example). This purpose, $p1$, might be inherited by "content recommendations," $p1.1$. Both would inform the data subject about the categories of personal data used, but these would not expose any further insights. However, by adding underlying purposes to $p1.1$, it is possible to elaborate the nature of "content recommendations" further. For instance, in Fig. 1 the purpose $p1.1$ composes "collect page view analytics" ($p2'$), "profile preferences" ($p3'$), and "personalize feeds" ($p4'$). Clearly, when compared to displaying only $p1.1$, these compositions make the policy more transparent to the data subject.

It is possible to construct a policy that does not meet the properties required, but it would not be *valid*. Using the definition of $up$ in (3), it is possible to display a privacy policy organized as a directed acyclic graph. A cyclic graph could be, theoretically, viable in a policy (as long as the constraints for valid policies are met), but we have not found meaningful use cases. The root elements of a policy are those which are not underlying any other purpose—and following the constraints, the data subject only needs to study and accept these. An interested party may read deeper. Furthermore, the formal definitions enable the subsequent functionality: coupling the purposes with technical functions.

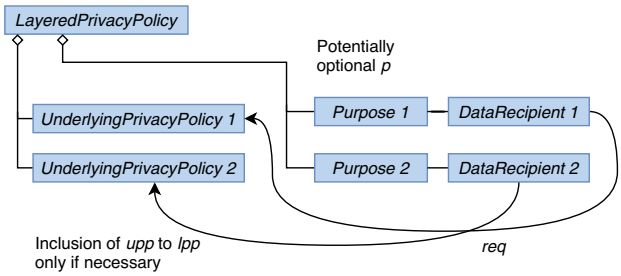


**Fig. 2** How the relationships of *UnderlyingPrivacyPolicies*, *DataRecipients*, and *Purposes* are structured

## 3.2 Data transfers in LPL

In addition to the structure of $Purposes$, we can also make observations about data transfers in LPL and suggest two improvements to facilitate policy creation in this regard. In the original form, only a single $upp$ can exist for an $lpp$, which enables chaining previously consented policies, and, therefore, data transfer to third parties. In practice, it appears more useful to treat the $upp$ of a policy as a set of $UnderlyingPrivacyPolicies$; a single policy could model transfers to multiple controllers, which are not necessarily related to each other. A situation of this kind is difficult to model clearly in the original structure with a single $upp$ and would result in multiple top-level policies that could be mostly redundant. The shorthand we propose, using a set of $upp$, appears to make modeling multiple third parties simpler and does not break any features of the language. It is not a solved issue whether you should combine policies from different parties into one when presenting it to the data subject. Our position is that while the arrangement might not be viable in practice at the present moment, it is an opportunity that the language could support.

Furthermore, it should be noted that $DataRecipients$, $upp$, and $Purposes$ are closely connected. Since a $Purpose$ $p$ can be optional and thus a data subject can reject it, the $\widehat{DR}_p$ of that $p$ can be ignored in the effective policy. If a rejected $p$ would have featured a data transfer, the $upp$ of the recipient is no longer required in the policy. There exists, then, an optional relationship between $DataRecipient$ $dr$ and $upp$. Therefore, let

$$req = (dr, upp), \tag{13}$$

where the existence of a pair $(dr, upp)$ in the set of $\widehat{REQ}$ denotes that the $dr$ requires the inclusion of a $upp$ to be used in a $Purpose$. How these constructions relate to each other is displayed in Fig. 2 as an example.

### 3.3 Web services as privacy policy purposes

The composed purpose specifications elaborated allow creating more informative privacy policies. These empower specifying down from an abstract level of a personal data processing *Purpose* to what the processing concretely consists of. These specifications enable a deeper validation of a policy, more transparency, confidence that the policy matches a corresponding technical implementation, and the subsequent static analysis method. Although the composed $Purpose$ is entirely domain-agnostic, it is particularly useful in the web application domain. Next, this section presents a way to attach the LPL to a model of a web service.

Consider the naïve definition of a web service as a function of requests to responses (see the definition of a server in RFC 2616; [22]). The web service function can be uniquely identified by a Uniform Resource Locator *(URL)* over the Web. To increase the abstraction depth, we can model what the web service function *does* using the concepts of web services and service nets (SNs) [30]. Roughly, a web service is a tuple of values describing the service, a set of component web services, and a service net. An SN, in turn, is a place-transition Petri net modeling the dynamic behavior of a given web service. To proceed more formally, let

$$WS = (NameS, Desc, Loc, URL, CS, SN), \tag{14}$$

where $NameS$ is the unique identification code of the service, $Desc$ a textual description for it, $Loc$ the server in which the service is located, $URL$ its endpoint, $CS$ a set of component web services, and $SN$ the service net describing its behavior [30]. A service net, in turn, can be defined as a tuple

$$SN = (P, T, W, i, o, \ell), \tag{15}$$

where $P$ is a set of *places*, $T$ a set of *transitions* representing the operations of the service, $W$ a set of directed *arcs*, $i$ the input place, $o$ the output place, and $\ell$ a labeling function for transitions [30]. A couple of assumptions allow to connect these definitions to personal data processing. First, in the present context, any web service is assumed to be governed by a privacy policy, and that policy can be modeled with LPL. Second, the dynamic behavior exposed by the Petri net model can be exploited by assuming that personal data processing occurs always within $T$. The latter assumption necessitates a more thorough inspection of the behavior of a service and assumes it is possible to do so.

For a simple instance, say that an $SN$ of a web service has an input place $i$, an output $o$, and three transitions with labels "*register customer*" ($t1$), "*create subscription*" ($t2$), and "*send confirmation email*" ($t3$). The set of arcs $W$ connect $i \rightarrow t1 \rightarrow t2 \rightarrow t3 \rightarrow o$. Each transition processes its own set of personal data. If this processing can be inspected, a union can be used to define the total processing for the given $SN$. The principle remains the same in a more complex instance with branches: A union is the sum of all potential transitions in the service net.

To this end, a function that inspects personal data processing in any given web service can be defined as:

$$pd : WS \rightarrow (\widehat{D}_{ws}, \widehat{DR}_{ws}, \widehat{UPP}_{ws}), \tag{16}$$

where $WS$ is a web service and the return value is a tuple describing data processing in its behavior. In this tuple, $\widehat{D}_{ws}$ is a set of personal data (i.e., LPL's $Data$ elements) processed in the transitions of a service net of the $WS$, $\widehat{DR}_{ws}$ is a set of authorized parties allowed to access the personal data (i.e., LPL's $DataRecipient$ elements), and, finally, $\widehat{UPP}_{ws}$ is a set of privacy policies (i.e., $UnderlyingPrivacyPolicy$ elements) of third parties the WS transfers personal data to.

The function $pd$ allows modeling personal data processing at a sufficiently abstract level. Also the mapping of the definition (14) to LPL $Purpose$ is straightforward by using the composition rules described earlier. In essence: (i) any $WS$ processing personal data is governed by a privacy policy; (ii) the act of *processing* in LPL is encoded in $Purposes$; and, therefore, (iii) at least one $Purpose$ governs any web service processing personal data. In other words, the definitions (2) and (14) are composable and flow from a high abstraction level to concrete operations. These can be further mapped to the definition (1) by noting that (iv) there exists a $Purpose$ for any web service $WS$ in a set of web services $\widehat{WS}$ governed by a *LayeredPrivacyPolicy*. In other words, for all $WS \in \widehat{WS}$ exists $p \in lpp$ for which

$$\begin{aligned} pd(WS) := (&\widehat{D}_{ws} \subseteq \widehat{D}, \\ &\widehat{DR}_{ws} \subseteq \widehat{DR}, \\ &\widehat{UPP}_{ws} \in upp_{lpp}). \end{aligned} \tag{17}$$

Finally, in order to simplify the notation, let the following tuple mark the relation between a $Purpose$ and a web service:

$$gov = (WS, p), \tag{18}$$

which also implies that

$$\widehat{D}_{ws} \subseteq \widehat{D}_p \text{ and } \widehat{DR}_{ws} \subseteq \widehat{DR}_p \tag{19}$$

for any $gov(WS, p)$. While the relation $gov$ is not specified as a part of a privacy policy, this mapping between the policy and a description of a service provides a useful method for data controllers. Although any compliant web service processing personal data does so under a privacy policy, this

condition does not dictate the practice. For instance, a single $Purpose$ could govern all web services of a policy, or a single $WS$ might be governed by multiple $Purposes$. In practice, it is also possible that a policy is *invalid*: A $pd(WS)$ does not match any $Purpose$ in a policy. Such cases should be avoided, obviously.

A further benefit originates from the fact that modeling can be done at either side of the abstraction; at the level of web services; or at the level of privacy policies. The presented method couples the two models together but leaves the responsibility of the desired abstraction to the data controller. Therefore, it is possible to "attach" a governed web service at any level of the LPL *Purpose* graph. In addition, composed web services can be modeled, which offers another dimension for flexibility. To illustrate these points, Fig. 3 shows a web service $S$ composed of two services $S1$ and $S2$, both of which have a governing $Purpose$. The $Data$ and $DataRecipient$ elements are derived from the $SN$ of the web service in question through the function $pd$. The depth and granularity of a privacy policy can be chosen as it is written. Different configurations and their privacy implications are discussed later on in Sect. 6.

## 4 Implementation of approach

The method of our approach has two steps. The first step is to use *static declarations* of personal data processing in annotations of source code. While in practice the exact construction is specific to a programming language, many languages support some way of encoding meta-data to function or class declarations. After injecting this meta-data is applied during programming, it can be leveraged in the next step, as a compile-time *static analysis* of the source code. Thus, we can reason about the structure of a program and the way the program processes personal data, and to use this information to create privacy policy language constructs from the capabilities of a given web service.

### 4.1 Annotations for personal data

The original toolkit for personal data annotations [33] featured three annotations: `@PersonalData` for classes and variables containing personal data, `@PersonalDataHandler` to declare areas of a code base which may process personal data, and `@PersonalDataEndpoint` to declare the boundaries of the analysis. In that work, the use case was validating source code structure. For the present work of extracting data for privacy policies, the annotation set is useful as a baseline, and we build upon the idea that information about data sensitivity can be embedded statically to source code. In what follows, the theoretical model already described is further refined into practical use. To account for personal data transfers, we introduce an additional annotation to the annotation toolkit.

### 4.2 Purposes

Since the personal data processing purposes coupled to web services are derived from the behavior of services, the next part is to define a way to generate data for privacy policies via static analysis. This construction requires limiting the scope to a specific implementation level. To still retain a high degree of generality, the practical solution discussed focuses on a particular but common design pattern for SOA web services.

In essence, many web frameworks use the same strategy of defining application entry-points in the source code by using annotations. Good examples include the Spring framework[5] for Java and Flask[6] for Python. In this pattern, server functionality is run by a given library, which maps incoming requests to (application code) function calls based on annotated entry-point definitions. Strictly speaking, the pattern is a composite web service, which conditionally calls a different component web service based on a given request. Considering it as a single service with multiple entry-points is a reasonable simplification; the source code for many entry-points may belong to a single file or class, for instance. In the present work, each entry-point maps to a privacy policy $Purpose$, as defined in Sect. 3.3.

Continuing with the annotation-based approach, the $Data$ elements can be derived from *object relational mapping* (ORM) classes. Many web service frameworks offer annotations to mark these declaratively and to define the relational mappings statically (see Hibernate[7], for instance). Personal data stored in these database entity objects can be reasoned about if the semantic information is provided in a similar way [33]. Obviously, personal data processing is not limited to database interactions, so our approach of using ORM-mappings is only a baseline standard. The relational table mappings provide a useful categorization of personal data in a code base, but there are no issues with expanding the analysis scope. For example, even if a message-passing web service had no database at all, one could apply the method to `@PersonalData`-annotated *data transfer objects*.

Using both the application entry-point and personal data ORM-annotations, a code base declares how personal data is processed statically. These declarations map to the theoretical service net definition in a concrete manner. By using static analysis processing, it is possible to create (at the least templates for) *Purpose* definitions. Since each entry-point is a "main" function of a subprogram of a web service, they form a distinct directed graph of function dependencies.

[5] https://spring.io/.

[6] https://pypi.org/project/Flask/.

[7] https://hibernate.org/orm/.

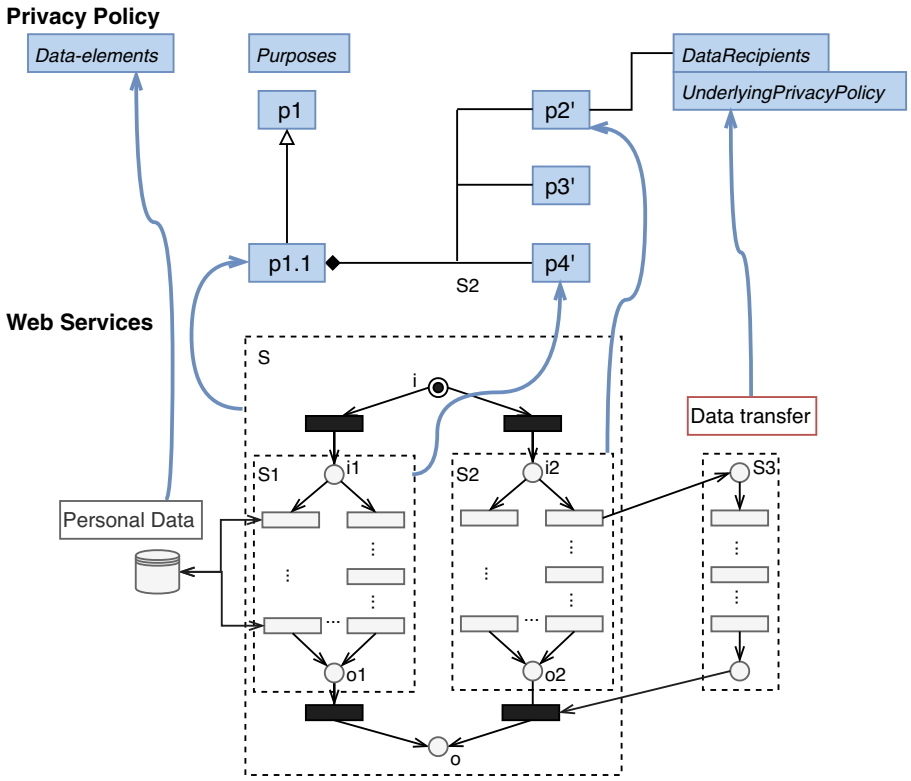


**Fig. 3** Illustration of LPL *Purposes* mapping to composed web services; thick arrows represent privacy policy data generated from web services. Extended from [30,34]

As the abstraction levels of the graph range from modules and classes to methods, the *Purposes* compose in a similar manner. Analyzing each function of the code base during compilation allows marking personal data a given function depends on.

The other values from the tuple $p$ in (2), including *name* and *descr*, can also be generated with static analysis and matching annotations. That said, the values for $optOut$, *required*, retention $r$, and privacy model $pm$ require human judgment in the present work. In addition, a web service could have capabilities beyond the reach of static analysis, thus confirming that the subsets of the mentioned elements are indeed complete sets—and completing them is also a manual process.

The method we have so far described addresses generating *Purpose* templates with *Data* elements from source code in a specific architecture. Full capabilities of the inspection function *pd* require providing the $DataRecipients$ of a web service—as well as the set of $UnderlyingPrivacyPolicies$ of third parties the $WS$ transfers data to.

### 4.3 Data transfers

Referring to the definition of *pd* in (16), a necessary dimension for generating privacy policies is detecting *transfers* of personal data to the realm of other policies. For any WS under inspection, the set of underlying privacy policies $\widehat{UPP}_{ws}$ must exist. Since any WS processing personal data are governed by a policy, it follows that any WS that receives data has also a policy of its own. The layered nature of the LPL implies that to accept a policy, one must accept all underlying policies contained within. Consider then $WS_1$ which calls $WS_2$ through a software development kit (SDK) library, transferring personal data in the process. In order to create the privacy policy governing $WS_1$, it is then required to find the policy of $WS_2$. If $WS_2$ has further transfers, those must be also located, and so forth.

Compiling all the layered privacy policies into a single one while developing the original $WS_1$ is impractical. They must be available at run time—otherwise a data subject cannot consent. To further complicate the process, it is not unimaginable to have circular dependencies in this stack of layered policies. To solve this problem, our approach simply links the underlying policies. The management of the links is then automated via the data extraction tool.

Managing the links to underlying privacy policies is accomplished using another annotation in the source code. We extend the personal data annotation toolkit [33] with `@PersonalDataTransfer`, which is applied to a function or a class. In addition to marking a point of a data transfer in source code statically, an annotation instance is parameterized to contain the identifier of the policy it refers to. Thus, the static analysis process for extracting purposes also has the capability to find all underlying policies used in a WS. How the web service structure can be mapped to a privacy policy is illustrated in Fig. 4.

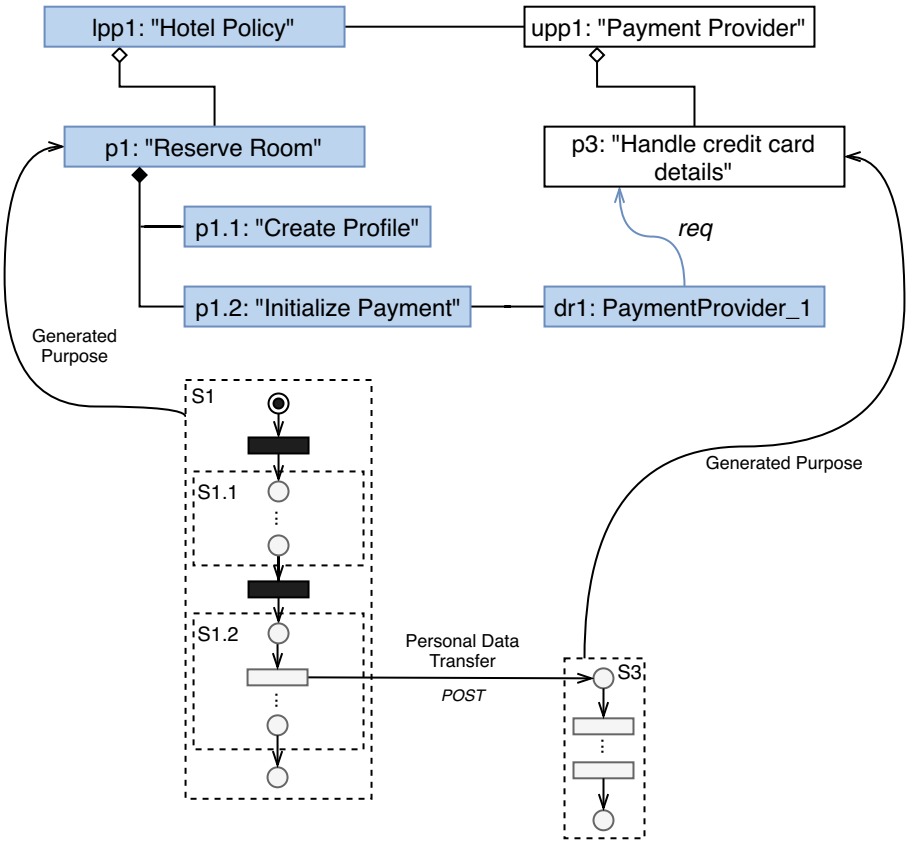


**Fig. 4** Illustration of personal data transfers mapping to first- and third-party privacy policies from Web Service *Service Nets*

The ideal use case and the optimal path for the spread of practical use of the `@PersonalDataTransfer` annotation would be SDK libraries that provide clients services with predefined annotations. With this assumption, correct linking occurs without effort from a developer. In practice, however, such optimal situation would require effort from a whole software ecosystem. Therefore, it remains a task for developers to annotate functions that perform data transfers in a given application code.

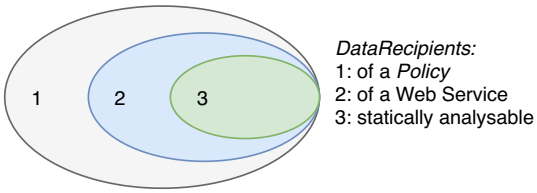


**Fig. 5** A Venn diagram illustrating the limits of the $Data-Recipients$ analysis

## 4.4 Data recipients

The mechanism for detecting occurrences of data transfers can also be applied to a class of $DataRecipients$. Referring again to definition of $pd$ in (16), and recalling the definition of $p$ in (2), to create a complete privacy policy and purposes for it, the framework ought to extract $DataRecipient$-elements from a WS. To be clear, our method detects only a subset of $\widehat{DR}_{ws}$; those associated with the transfer of personal data. Other groups would include different classes of processors of the data controller of the policy in question. This requirement is partially expressed in definition (17), where any WS inspected by $pd(WS)$ is a subset of the $DataRecipients$ of the privacy policy. The limits are further illustrated in Fig. 5.

In LPL, $DataRecipients$ exist as one of the components of a $Purpose$. They are defined as *"the authority that gets specific processing rights (defined by the Purpose) granted"* [25] by a $Purpose$. In our context of technical $Purposes$ we are extracting from a code base, $DataRecipients$ represent those entities the web service transfers data to, during the execution of one of the entry-point functions. To find this information through static analysis, we can refer to the strategy for analyzing data transfers. The issue amounts to encoding $DataRecipient$-information to the source code annotations in a suitable and sustainable way. To avoid redundancy, our approach again only encodes the identifier of a $DataRecipient$ (*name*-field in LPL) in the parameters of the `@PersonalDataTransfer` annotation. Thus, data transfers and $DataRecipients$ are tightly coupled in our

approach, and the relationship $req(dr, upp)$ in (13) is in the scope of the analysis.

### 4.5 Implementation as data extraction tool

To demonstrate practical use, a concrete implementation was developed for the Java programming language. The data extraction part of the program was developed as a compiler plugin for the Java language. The tool runs as an annotation processing tool (APT) and generates a fresh dataset automatically upon building a project it is enabled on. The source code of the tool is published under an open source license.[8] Using the APT interface, the package can be integrated to both command line tools and integrated development environments. In addition, a data visualization script was developed during the study.

The prototype implementation is limited to a particular set of supported annotations using the Spring framework: `@RequestMapping` and `@Document`. These annotations are used, by framework convention, to mark application entry-point functions and ORM-classes, respectively. In addition to these, the implementation requires the use of the personal data annotations described earlier. It should be relatively trivial to extend the support for alternative framework annotations, though, as long as the target architecture is similar.

Developing the inspection process for the specific environment required some further design choices. Given that there are no right or wrong answers in this context, other implementations might differ slightly. In particular, Java's Interfaces are handled with a pessimistic approach. For instance, consider a $Purpose$ entry-point using an interface with multiple implementations. The tool implemented handles this by summing the personal data found in all available implementations. Another design choice made pertains to other modules (libraries) imported in a web service. As the tool is tied to compile-time static analysis, other (precompiled) modules are out of the scope of the tool. In practice, this choice was not seen as a hindrance because the given business logic generally defines the type of personal data processed. Regarding data transfers, the convention for libraries is to declare `@PersonalDataTransfer` in the interface provided.

To best of our knowledge, no standard data file format for LPL exists. While different use cases have different needs, a uniform format could be beneficial for the adoption of LPL. This task is out of the scope of this paper, however. For the purposes of the tool presented, the extracted $Purposes$ are stored as JavaScript Object Notation (JSON) files. The natural mapping is using the names of the LPL tuple values as keys in the objects themselves. Our data format also omits the contents of $Data$-elements and refers to them by data group name. These decisions are mostly incidental and serve practical needs; should a data format become standard, the method will adapt. To summarize, the tool presented implements the data extraction needs for a specific case, of the general solution that was outlined.

[8] https://github.com/devgeniem/personaldataflow.

## 5 Validation: a case study

To demonstrate the presented method in practical use, the implemented tool was applied to a web service code base of a company. The goal of the case study was to validate that the static analysis method and the developed tool both work as intended. The case study methodology was simple: The generated documentation should match the expectations of the implicit model of the case web service. This was compared by hand, matching the data with the API specification and usage of the case service—this process did not require interpretation as the generated data must match the source code exactly. Building the final policy would be subjective, and thus not the focus of this work. The initial analysis of the case web service was done in [34], and the data extraction and analysis were repeated after the extensions presented in this paper, in order to have the full set of policy data. As the code base had slightly evolved in-between, the numbers presented here are partially different from the initial work.

The case $WS$ has been in production for processing transactions in a Finnish company since 2014. There have been different maintainers and product owners. Despite ongoing maintenance and development, there was no tooling for automated documentation in use before the case study. Deploying the tool required annotating the personal data database entity classes and installing the annotation processor to the build chain. This integration amounted to a moderate amount of work. The moderate work amount indicates applicability of the tool to new targets with roughly similar software architectures. After the initial setup, repeated data extraction and tool version updates were low effort. To summarize, integrating the tool into the code base was successful.

The case $WS$ is a single monolithic code base, which is split into modules. Despite having multiple responsibilities, the $WS$ uses and is used by multiple service. The total of 323 classes (amounting to about 28 thousand lines of code) at the time of the initial analysis can be understood as a *Model/View/Controller* architecture with specific service and database layers. Considering the definition for composable web services, the entire system would be the root $WS$. It would compose $WS$ modules ("Controllers"), which, in turn, compose $WS$ endpoint functions. These functions are governed by composed $Purposes$, which the data extraction tool maps. Although the case $WS$ uses several libraries, these being out of scope were not a hindrance for analysis; as dis-

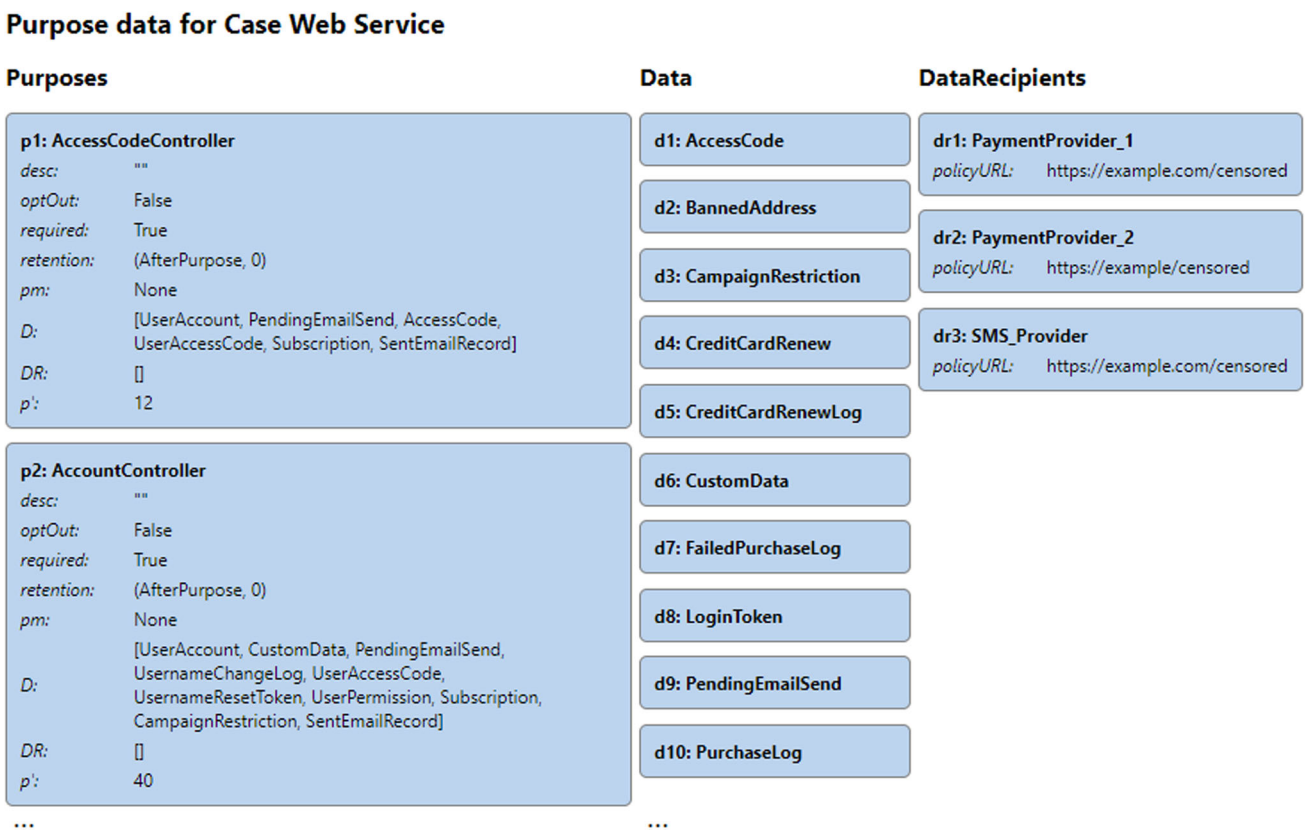


**Fig. 6** Excerpt of Purpose data visualization generated from the case WS, featuring Purposes, Data-elements, and DataRecipients. In the Purposes, *p'* stands for the number of sub-purposes

cussed in Sect. 4.5, any library processing personal data are supplied the data by the business logic code that is in the scope of the analysis. It should also be noted that authorization for the entry-points is out of scope of the analysis; the sessions are checked in the framework rather than in the code inspected.

The results of the data extracted by the tool can be summarized as follows. The case $WS$ was found to comprise 30 Controller modules, each of which was composed of multiple entry-points (a total of 245 with a range of [1, 51], and an average of 8.2). Out of these, not all processed personal data. Regarding those that concern privacy policies, a total of 24 Controller modules remained (with a total of 224, a range of [1, 51], and an average 9.3). In total, 19 different personal data entity and three $DataRecipient$ types were identified. Each controller processed on average three personal data classes with a range of [1, 11].

Following our formulation of the problem, the case $WS$ was viewed as set of component web services (i.e., Controller modules), which in turn are composed of entry-point web services. The data visualization tool developed as a part of the effort is demonstrated in Figs. 6 and 7, which display generated figures of the different types of policy data extracted (after slight post-processing: cropping, censoring private titles). The analysis tool extracts the data for each entry-point, most of which were omitted from these figures for brevity. As defined before, the sets of $Data$ elements belonging to any component $Purposes$ $p'$ are subsets of their higher-level $Purpose$ $p$. The data also visualize the application architecture and highlight important areas: those where many entities concentrate in a single $Purpose$ as well those where entities are used across multiple purposes.

There is no existing formal privacy policy for the case $WS$, so an exact number for coverage cannot be given. Regardless, it is possible to gain insights by comparing the implicit $Purposes$ in the system to those extracted with the tool implemented. With manual verification, it is confirmed that the results of the analysis cover all of the entry-points. Thus, in this sense, the tool appears sound. However, the complete logical hierarchy of the composed $Purposes$ cannot be constructed by inspecting the $WS$ source code alone. As a counterexample, the process of purchasing (a purpose) composes of entry-point $Purposes$ from web service's $SubscriptionController$ and $LoginController$. This purpose can only be inferred by viewing the clients in addition to the web service. An example of this case is illustrated in Fig. 8, where composed $Purposes$ $(p22, p175')$ and

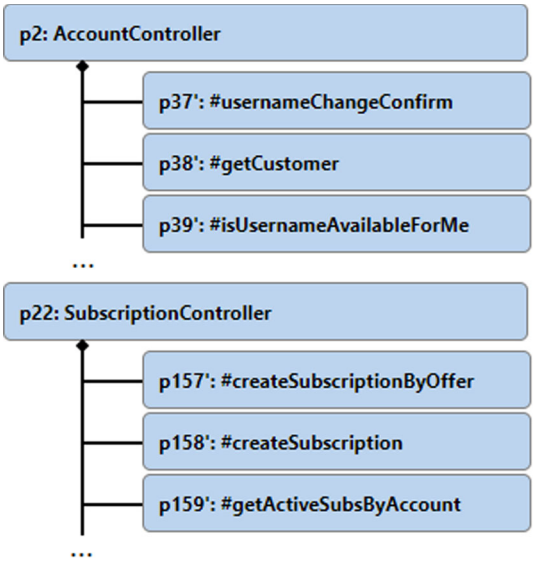


**Fig. 7** Excerpt of *ComposedPurposes* data visualization generated from the case WS

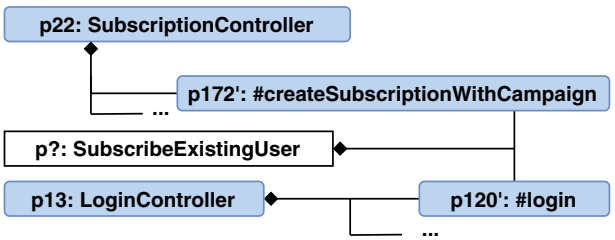


**Fig. 8** Another example of composed *Purposes* of the case study, with an identified out of analysis scope *Purpose* (white) inferred from client-side. Extended from [34]

($p13$, $p120'$) also have a third *Purpose* $p?$ that could not be found in the web service analysis.

The case study demonstrates the possibility to construct any *Purpose* concerning the case system, by composing the extracted *Purposes* that act as building blocks. The presented method thus answers to RQ2. To create the whole *Policy*, it is not sufficient to analyze server-side code alone. As the web service API provides a collection of capabilities, different possible combinations of these cannot be constructed without additional context.

## 6 Discussion

This paper provides a method for improved means to express and reason about privacy policies and personal data processing formally. In essence, any web service processing personal data must be governed by a privacy policy. The method presented in this paper does not, however, *enforce* that transparency for a data subject is increased. Having the means to automatically generate data might lower the effort of creating expressive privacy policies. The results of this paper still assume a good faith effort on part of the service provider; maintaining a policy with more detail could also be a liability for maintenance. Moreover, a vague policy that is decoupled from the technical means of the service does allow more "freedom" for the data controller. The question is then: Is it worthwhile?

As discussed in the motivation of the paper, a privacy policy must be specific and explicit [42]. The first principle of the GPDR requires personal data processing to be lawful, fair, and transparent [54]. In a practical environment, these abstract principles are weighed against multiple other constraints and business requirements [32]. The example and standards put forth by the privacy engineering research community is a way to drive industry adoption of privacy-friendly means for data processing. This paper is a small step on that path, which also requires cultural changes in addition to technical means. Needless to say, change is not easy. The automated method to generate privacy policy data does require a change in practice instead of only implementing technical improvements—but the process can be incremental and iterative.

The question can be raised whether using source code as the basis for generating privacy policy data is useful in practice, in comparison with other approaches. Another alternative that has been explored, see Sect. 2, would be annotating Business Process Models instead and using that for static analysis. There are trade-offs between these approaches. Using source code, as per our approach, maintains a single source of truth for the privacy policy data and the model will not become out of sync with reality. Admittedly, this way of relinquishing control of policy to developers is not suitable for every organization, and thus, adopting this method requires consideration.

The presented method for automatically generated *Purposes* cannot fully describe a privacy policy in most cases, as mentioned in Sect. 5. This result can be viewed through an optimistic lens, remarking that fully generated privacy policies are not realistically *wanted* to begin with. Indeed, some authors have recently warned about using automatically generated privacy policies without prior consultation with legal experts [53]. That said, tools still make the task easier; the method is suitable for generating building blocks from which the final policy is compiled from. Rarely does the web services hierarchy of a business, which can be modeled, exhaustively map to the logical processes used by clients. However, as we can claim, any policy that uses a web service ought to include the *Purpose* governing that *WS* in its composition. The minimum viable policy would be one *Purpose* which governs all web services, but there is no upper limit (other than practicality) to the number of *Purposes* governing any given web service. These assumptions raise the question whether it is possible to draw any

judgments about the quality of a privacy policy by looking at the LPL structure.

Even looking at a *valid* LPL policy, it is not guaranteed that it is *specific* or *explicit* enough to be compliant with the GDPR. As we are mapping (composed) *Purposes* to web service entry-points, it would seem that a single *Purpose* to a single *WS* entry-point is the most explicit mapping that can be constructed. Having a single *Purpose* governing many *WS* entry-points could be made more explicit by detailing the purpose of each entry-point. A single *WS* entry-point serving multiple purposes—besides being questionable programming practice—could be split to multiple entry-points with specific functionality. It should be noted that these considerations apply to the lowest-level composable *Purposes*, and the more abstract high-level *Purposes* have their own scope.

The method of this paper then suggests a *ground-up* approach to arranging a privacy policy. If the lowest composed purposes layer follows the one-to-one mapping with technical functionality, the higher abstraction-level purposes based on business needs are built on specific and explicit ground. Building on from these assumptions, the breadth of the *ComposedPurposes* graph should be minimized (i.e., consent is not asked for unnecessary purposes), while the height appears only as the consequence of raising abstraction level and bundling low-level *Purposes*.

The quality of a privacy policy still has nuances. For instance, it cannot be conjectured that more information would always imply "better" privacy policies [9]. A privacy policy that overwhelms a data subject with information might not be presented in "*intelligible and easily accessible form, using clear and plain language*" [54]. By implication, the legal requirements for a consent might not be satisfied. The composed purpose definition of this paper somewhat alleviates this issue: It is possible to specify purposes in detail and only show deeper levels of a given tree to those data subjects who are interested in the details. To overcome potentially confusing and ambiguous linguistic expressions, standardized terms and expressions could be used for *Purpose* descriptions. Also graphical user interface elements and other visual cues could be used to improve the presentation. While our approach leaves the final composition of a privacy policy to a practitioner, the automatic data generation facilitates a fine-grained policy.

### 6.1 Integration in engineering practice

We propose three ways to make use of the data extraction tool in software engineering practice and present a reference architecture to demonstrate these use cases. The *Purpose* data extracted from a code base serve different needs in different phases of development. We refer to these three as *activities*.

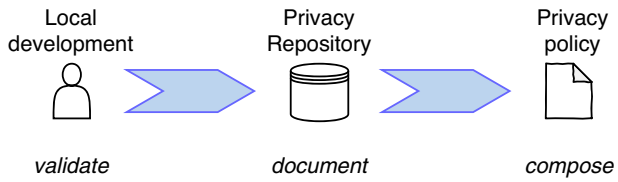


**Fig. 9** Proposed use-case activities for the data extraction tool

Before continuing, a few assumptions must be made about the context. Let there be a process of continuous development of multiple web services in a business with multiple teams. Here, continuous means that there is no single project deliverable, but iterations over an undefined time period. Multiple teams and web services imply that there are multiple privacy policies, which may possibly overlap. Continuous development, multiple teams, and many web services all necessitate internal documentation even in agile environments [6].

The first activity for the data extraction tool is *validation* during development. As a developer compiles code, the tool can be used to extract the current purposes matching current version of the code. The results can either be checked with unit tests (comparing to data from previous versions) or reviewed manually that they match expectations. The use case here is to validate that code changes do not process personal data in an unexpected way.

After local changes to source code are integrated to a main repository in a continuous integration or related system, the tool can *document Purpose* data in an internal "privacy repository." We propose this documentation as an integrated part of an enterprise meta-data management strategy (see [16] for an example). The *Purposes* are stored as-is. They also represent capabilities of a web service. If required, the repository allows to further enrich the data with values not created by the tool. For instance, additional *Purposes* not within the realm of static analysis can be created with other means. To facilitate this use case, the data extraction tool repository features a visualization script that generates hypertext markup language (HTML) documents from the *Purpose* data.

Finally, the data collected in the privacy repository can be used to *compose privacy policies*. Since these documents have legal implications, it seems useful to separate creating them from the data extraction. The *Purposes* (and other collected data) are then used as building blocks for the policy. At this step, the policies from third parties must be also compiled into the final product.

The proposed process and the activities are summarized in Fig. 9. Obviously, there may be also other use cases beyond the three described. To this end, the use cases could be used as the starting point for developing mature industry-wide solutions.

### 6.2 Limitations and further work

The presented work is by no means exhaustive. Leveraging static analysis for privacy policies has further potential, and the presented work is limited in many aspects. Some limitations of this paper are a deliberate scoping choice, which can be briefly discussed further.

Our approach leverages static information encoded in source code files for generating the privacy policy $Purposes$ of web services. To achieve this, there are some costs for the practitioner in the form of additional work in setup and development. To generate the data using the architecture described in Sect. 6.1, a practitioner must (1) *annotate source code*, (2) *maintain data extraction tool infrastructure*, and (3) *compile the final privacy policies from the data*. In comparison, a lightweight *privacy-by-policy* approach must only maintain incremental versions of a textual privacy policy. While the crux for adoption covers all three facets, these are all essentially context-dependent. First, whether automatically updating data based on software updates is useful. Second, whether machine-readable privacy policy data is useful. And last, how is the transparency benefit of more fine-grained policies valued. One argument to support these qualities is the value of having internal documentation in addition to customer-facing policies.

The definition of $ComposedPurposes$ has the necessary limitation that a sub-purpose cannot have more lax privacy attributes than its parent. This work has deliberately limited its scope so that we do not consider the "privacy hierarchy" of different anonymization methods and instead require sub-purposes to implement the same method as the parent. Should some future work present an unambiguous hierarchy of the methods that defines comparison between them; the restriction imposed by us in (7) can be reduced accordingly.

The implementation of $pd$ within the constraints defined in Sect. 4 is just one instance of the general problem. However, similar definitions should be possible for other architectures and programming languages. Dynamically typed environments would of course require another approach for the analysis part. While the analysis method does cover data transfers, to gain full extent of the value of LPL also the recipient parties should describe their processing in a formal language. At this time, translations between different privacy languages are not developed (which is a potential research space). In practice, this implies that to create a full LPL policy, the data recipients should also use LPL. Extending the method to further programming languages is more challenging, but not impossible. As of yet, only the data extraction part is fully automatic: Visualizing the results in an IDE plugin is a possible opportunity for further work.

As discussed previously, our approach of generating $Purposes$ from web services does not have the full information for a complete $Policy$ on its own. In addition to the $Purposes$ matching web services, a complete $ComposedPurposes$ graph requires client-side information to know the combinations on how the building blocks are arranged. Further research might analyze client-side source code in order to combine data with the web service $Purposes$. Although a similar approach would suffice, the method would require novelty: Whereas the execution tree of a web service starts from a single point and branches deep, client-side code typically has a wide array of different execution paths that culminate into web service calls. Completely removing human judgment from the process of creating privacy policies might not even be useful. If generated data are used as a basis to build a policy manually, more coverage always provides more options.

The case study presented in this work should be considered merely a validation project. While an example was presented on how the method works as a proof of concept, more comprehensive experiments with multiple different code bases are needed for quantitative evaluation. Likewise, qualitative evaluation of the approach and the tool is necessary as well. This paper proposed some ways to integrate the data extraction into the software engineering process, and that was not an exhaustive example at all. For instance, a better theory around what constitutes a "good" arrangement of $Purposes$ to web services could be studied. Another path forward would be to integrate LPL (or another related language) to the OpenAPI standard (or a similar specification) and generate the corresponding documents automatically. This path would fully start to leverage formal privacy languages in the semantic web.

### 6.3 Conclusion

This paper sought answers to two research questions. The first asked about a way to formalize an LPL model to describe personal data processing in web services. To this end, the privacy language was extended to include composition for $Purposes$. This extension was formally integrated with the definition of web services to form the model that couples privacy policies and web services together.

The second question solicited a method for extracting the corresponding data automatically from a web service code base. This question was approached both theoretically and practically. In theory, by defining an inspection function that would satisfy $LPL$ requirements of both $Purposes$ and data transfers. In practice, limiting the work to a certain web service design pattern: by presenting and evaluating an analysis tool for the method. The results were validated with a case study.

**Acknowledgements** We thank the three anonymous reviewers whose comments have improved this manuscript.

**Funding** Open Access funding provided by University of Turku (UTU) including Turku University Central Hospital. This research was partially supported by the Strategic Research Council at the Academy of Finland (grant number 327391) and Geniem Oy.



## References

1. Aktug, I., Naliuka, K.: ConSpec—a formal language for policy specification. Sci. Comput. Program. **74**(1–2), 2–12 (2008)
2. Amato, F., Coppolino, L., D'Antonio, S., Mazzocca, N., Mosca, F., Sgaglione, L.: An abstract reasoning architecture for privacy policies monitoring. Futur. Gener. Comput. Syst. **106**, 393–400 (2020)
3. Ardagna, C.A., De Capitani di Vimercati, S., Paraboschi, S., Pedrini, E., Samarati, P.: An XACML-based privacy-centered access control system. In: Proceedings of the first ACM workshop on Information security governance, pp. 49–58 (2009)
4. Assembly, U.G.: Universal declaration of human rights. UN General Assembly **302**(2), 14–25 (1948)
5. Ayala-Rivera, V., Pasquale, L.: The grace period has ended: An approach to operationalize GDPR requirements. In: 2018 IEEE 26th International Requirements Engineering Conference (RE), pp. 136–146. IEEE (2018)
6. Baca, D., Carlsson, B.: Agile development with security engineering activities. In: Proceedings of the 2011 International Conference on Software and Systems Process, ICSSP '11, p. 149-158. Association for Computing Machinery (2011)
7. Bashir, M., Hayes, C., Lambert, A.D., Kesan, J.P.: Online privacy and informed consent: the dilemma of information asymmetry. Proc. Assoc. Inf. Sci. Technol. **52**(1), 1–10 (2015)
8. Basin, D., Debois, S., Hildebrandt, T.: On purpose and by necessity: compliance under the GDPR. In: International Conference on Financial Cryptography and Data Security, pp. 20–37. Springer, Berlin (2018)
9. Bechmann, A.: Non-informed consent cultures: privacy policies and app contracts on facebook. J. Media Bus. Stud. **11**(1), 21–38 (2014)
10. Bednar, K., Spiekermann, S., Langheinrich, M.: Engineering privacy by design: are engineers ready to live up to the challenge? Inf. Soc. **35**(3), 122–142 (2019)
11. Belli, L., Schwartz, M., Louzada, L.: Selling your soul while negotiating the conditions: from notice and consent to data control by design. Heal. Technol. **7**, 453–467 (2017)
12. Bieker, F., Friedewald, M., Hansen, M., Obersteller, H., Rost, M.: A process for data protection impact assessment under the European general data protection regulation. In: Annual Privacy Forum, pp. 21–37. Springer, Cham (2016)
13. Blanco-Lainé, G., Sottet, J.S., Dupuy-Chessa, S.: Using an enterprise architecture model for GDPR compliance principles. In: IFIP Working Conference on The Practice of Enterprise Modeling, pp. 199–214. Springer, Cham (2019)
14. Bonatti, P.A., Kirrane, S., Petrova, I.M., Sauro, L.: Machine understandable policies and GDPR compliance checking (2020). arxiv:2001.08930
15. Cranor, L.: Web privacy with P3P. "O'Reilly Media, Inc." (2002)
16. Dahlberg, T., Nokkala, T.: A framework for the corporate governance of data-theoretical background and empirical evidence. Bus. Manag. Educ. **13**(1), 25–45 (2015)
17. Dastgheib, S., Whetzel, T., Zaveri, A., Afrasiabe, C., Assis, P., Availlach, P., Jagodnik, K., Korodi, G., Pilarczyk, M., De Pons, J., et al.: The smartAPI ecosystem for making web APIs fair. In: ISWC2017, the 16e International Semantic Web Conference **1931**, 1–4 (2017)
18. Degeling, M., Utz, C., Lentzsch, C., Hosseini, H., Schaub, F., Holz, T.: We value your privacy... now take some cookies: Measuring the GDPR's impact on web privacy. In: 26th Annual Network and Distributed System Security Symposium, NDSS. The Internet Society (2019)
19. Diver, L., Schafer, B.: Opening the black box: Petri nets and privacy by design. Int. Rev. Law Comput. Technol. **31**(1), 68–90 (2017)
20. Ed-Douibi, H., Izquierdo, J.L.C., Cabot, J.: Example-driven web API specification discovery. In: European Conference on Modelling Foundations and Applications, pp. 267–28. Springer, Cham (2017)
21. Enck, W., Gilbert, P., Han, S., Tendulkar, V., Chun, B.G., Cox, L.P., Jung, J., McDaniel, P., Sheth, A.N.: TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. ACM Trans. Comput. Syst. (TOCS) **32**(2), 1–29 (2014)
22. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: RFC 2616: Hypertext transfer protocol–HTTP/1.1 (1999)
23. Fouad, I., Santos, C., Al Kassar, F., Bielova, N., Calzavara, S.: On compliance of cookie purposes with the purpose specification principle. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), pp. 326–333 (2020)
24. Gerl, A.: Extending layered privacy language to support privacy icons for a personal privacy policy user interface. In: Proceedings of the 32nd International BCS Human Computer Interaction Conference, p. 177. BCS Learning & Development Ltd. (2018)
25. Gerl, A.: Modelling of a privacy language and efficient policy-based de-identification. Ph.D. thesis, Université de Lyon; Universität Passau (Deutscheland) (2019)
26. Gerl, A., Bennani, N., Kosch, H., Brunie, L.: LPL, Towards a GDPR-Compliant Privacy Language: Formal Definition and Usage. Transactions on Large-Scale Data- and Knowledge-Centered Systems XXXVII (2018)
27. Gerl, A., Meier, B., Becher, S.: Let users control their data–privacy policy-based user interface design. In: International Conference on Human Interaction and Emerging Technologies, pp. 790–795. Springer, Cham (2019)
28. Giannopoulou, A.: Algorithmic systems: the consent is in the detail?. Internet Policy Rev. **9**(1) (2020)
29. Gürses, S., del Alamo, J.M.: Privacy engineering: shaping an emerging field of research and practice. IEEE Secur. Privacy **14**(2), 40–46 (2016)
30. Hamadi, R., Benatallah, B.: A Petri net-based model for web service composition. In: Proceedings of the 14th Australasian database conference-Volume 17, pp. 191–200. Australian Computer Society, Inc. (2003)
31. Hirsch, D.D.: The glass house effect: Big Data, the new oil, and the power of analogy. Me. L. Rev. **66**, 373 (2013)
32. Hjerppe, K., Ruohonen, J., Leppänen, V.: The general data protection regulation: requirements, architectures, and constraints. In: 2019 IEEE 27th International Requirements Engineering Conference (RE), pp. 265–275. IEEE (2019)

33. Hjerppe, K., Ruohonen, J., Leppänen, V.: Annotation-based static analysis for personal data protection. In: IFIP International Summer School on Privacy and Identity Management, pp. 343–358. Springer, Cham (2019)
34. Hjerppe, K., Ruohonen, J., Leppänen, V.: Extracting layered privacy language purposes from web services. In: 2020 IEEE European Symposium on Security and Privacy Workshops (EuroS PW), pp. 318–325 (2020)
35. Huth, D., Tanakol, A., Matthes, F.: Using enterprise architecture models for creating the record of processing activities (Art. 30 GDPR). In: 2019 IEEE 23rd International Enterprise Distributed Object Computing Conference (EDOC), pp. 98–104. IEEE (2019)
36. Khurat, A., Suntisrivaraporn, B., Gollmann, D.: Privacy policies verification in composite services using owl. Comput. Secur. **67**, 122–141 (2017)
37. Leicht, J., Gerl, A., Heisel, M.: Technical Report on the Extension of the Layered Privacy Language. Tech. rep. (2021)
38. Libert, T.: An automated approach to auditing disclosure of third-party data collection in website privacy policies. In: Proceedings of the World Wide Web Conference (WWW 2018), pp. 207–216. International World Wide Web Conferences Steering Committee (2018)
39. Lin, L., Hu, J., Zhang, J.: Packet: a privacy-aware access control policy composition method for services composition in cloud environments. Front. Comp. Sci. **10**(6), 1142–1157 (2016)
40. Martin, R.C., Newkirk, J., Koss, R.S.: Agile software development: principles, patterns, and practices, vol. 2. Prentice Hall Upper Saddle River, NJ (2003)
41. Martin, Y.S., Kung, A.: Methods and tools for GDPR compliance through privacy and data protection engineering. In: 2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), pp. 108–111. IEEE (2018)
42. Matte, C., Santos, C., Bielova, N.: Purposes in IAB Europe's TCF: which legal basis and how are they used by advertisers? In: Annual Privacy Forum (APF 2020) (2020)
43. Ramadan, Q., Strüber, D., Salnitri, M., Jürjens, J., Riediger, V., Staab, S.: A semi-automated BPMN-based framework for detecting conflicts between security, data-minimization, and fairness requirements. Software and Systems Modeling pp. 1–37 (2020)
44. Ramezanifarkhani, T., Owe, O., Tokas, S.: A secrecy-preserving language for distributed and object-oriented systems. J. Logic. Algebr. Methods Program. **99**, 1–25 (2018)
45. Ringmann, S.D., Langweg, H., Waldvogel, M.: Requirements for legally compliant software based on the GDPR. In: OTM Confederated International Conferences. On the Move to Meaningful Internet Systems, pp. 258–276. Springer, Cham (2018)
46. Sadeghi, A., Bagheri, H., Garcia, J., Malek, S.: A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. IEEE Trans. Softw. Eng. **43**(6), 492–530 (2016)
47. Salnitri, M., Angelopoulos, K., Pavlidis, M., Diamantopoulou, V., Mouratidis, H., Giorgini, P.: Modelling the interplay of security, privacy and trust in sociotechnical systems: a computer-aided design approach. Softw. Syst. Model. **19**(2), 467–491 (2020)
48. Senarath, A.R., Arachchilage, N.A.G.: Understanding user privacy expectations: a software developer's perspective. Telemat. Inform. **35**(70), 1845–1862 (2018)
49. Soh, S.Y.: Privacy nudges: an alternative regulatory mechanism to informed consent for online data protection behaviour. Eur. Data Prot. L. Rev. **5**, 65 (2019)
50. Spiekermann, S., Cranor, L.F.: Engineering privacy. IEEE Trans. Softw. Eng. **35**(1), 67–82 (2009)
51. Spiekermann, S., Novotny, A.: A vision for global privacy bridges: technical and legal measures for international data markets. Comput. Law Secur. Rev. **31**(2), 181–200 (2015)
52. Standard, OASIS: Extensible access control markup language (XACML) version 3.0 (2013)
53. Sun, R., Xue, M.: Quality assessment of online automated privacy policy generators: An empirical study. In: Proceedings of the Evaluation and Assessment in Software Engineering (EASE 2020), pp. 270–275. ACM (2020)
54. The European Union: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the Protection of Natural Persons with Regard to the Processing of Personal Data and on the Free Movement of Such Data, and Repealing Directive 95/46/EC (General Data Protection Regulation) (2016)
55. Tómasdóttir, K.F., Aniche, M., Van Deursen, A.: The adoption of javascript linters in practice: a case study on eslint. IEEE Trans. Softw. Eng. **46**, 863–891 (2018)
56. Tumer, A., Dogac, A., Toroslu, I.H.: Semantic-based user privacy protection framework for web services. In: Intelligent Techniques for Web Personalization (ITWP), pp. 289–305. Springer, Cham (2005)
57. Van Alsenoy, B.: Data protection law in the EU: roles, responsibilities and liability. J. Data Prot. Privacy **3**(1), 113–115 (2019)
58. Vanezi, E., Kouzapas, D., Kapitsaki, G.M., Philippou, A.: Towards GDPR compliant software design: A formal framework for analyzing system models. In: International Conference on Evaluation of Novel Approaches to Software Engineering, pp. 135–162. Springer, Cham (2019)
59. Verborgh, R., Harth, A., Maleshkova, M., Stadtmüller, S., Steiner, T., Taheriyan, M., Van de Walle, R.: Survey of semantic description of REST APIs. In: REST: Advanced Research Topics and Practical Applications, pp. 69–89. Springer, New York (2014)
60. Weitzner, D.J., Hendler, J., Berners-Lee, T., Connolly, D.: Creating a policy-aware web: Discretionary, rule-based access for the world wide web. In: Web and information security, pp. 1–31. IGI Global (2006)
61. Wilhelm, S., Gerl, A.: Policy-based authentication and authorization based on the layered privacy language. BTW 2019–Workshopband (2019)
62. Yan, D., Tian, Y.: Privacy policy composition of privacy-aware RBAC model for composite web services. In: 2013 5th IEEE International Conference on Broadband Network Multimedia Technology, pp. 312–316 (2013). https://doi.org/10.1109/ICBNMT.2013.6823964
63. Zaveri, A., Dastgheib, S., Wu, C., Whetzel, T., Verborgh, R., Avillach, P., Korodi, G., Terryn, R., Jagodnik, K., Assis, P., et al.: SmartAPI: Towards a more intelligent network of Web APIs. In: European Semantic Web Conference, pp. 154–169. Springer, Cham (2017)
64. Zhang, F., Fan, X., Zhou, W., Zhou, P.: Purpose-based access policy on provenance and data algebra. arXiv preprint arXiv:1912.00445 (2019)

**Kalle Hjerppe** is a doctoral student and a project researcher at the University of Turku. His research interest is in data protection and privacy engineering, intersecting with software engineering and architecture.

**Jukka Ruohonen** is currently working as a project researcher at the University of Turku. He has recently published in diverse journals, such as Information and Software Technology, Information Systems, Computers in Human Behavior, and Applied Computing and Informatics.

**Ville Leppänen** is a full professor in software engineering and software security in University of Turku, and the vice dean of Faculty of Technology. He has now over 230 international conference and journal publications, with research interests related broadly to software engineering and security, ranging from software engineering methodologies, practices, and tools to security and quality issues, as well as to programming languages, parallelism, and architectural design topics.